Modeling Adaptable Multimedia and Self-modifying Protocol Execution

Sheng-Uei Guan[1] and Sok-Seng Lim

Department of Electrical & Computer Engineering

National University of Singapore

Abstract

Over the years, researchers have tried to extend Petri net to model multimedia. The focus of the research flows from the synchronization of multimedia without user interactions, to interactions in distributed environments. The issues in concern are the flexibility and compactness of the model when applied to model a system under change. Most existing models lack the power to model a system under change during execution. Petri net extensions have been developed to facilitate user interactions (UI) in distributed environments, however they require sophisticated pre-planning to lay out detailed schedule changes. On the other hand, there has been active research on self-modifying protocols or adaptive protocols in recent years. Plenty of models have been developed to model communication protocol execution, to name a few, finite state machines, communicating finite state machines, Petri nets. However, there exist no suitable models to simulate protocols that are self-modifying or adaptive during execution. In this paper, we propose a Reconfigurable Petri Net (RPN) for adaptable multimedia. A RPN comprises of a novel mechanism called modifier. This modifier can create a new change or delete an existing mechanism (e.g. arc, place, token, transition, etc.) of the net. In a way, modifier embraces controllability, reconfigurability, and programmability into the Petri net, and enhances the real-time adaptive modeling power. This development allows a RPN to have a greater modeling power over other extended Petri nets. The paper includes both the model and theory required to establish the technique's validity. Examples are also shown how RPN can be used to model interactive multimedia, and simulate self-modifying protocols. A simulator has been developed using Visual C++ under Windows NT to show that RPN is feasible.

[1] Corresponding author

Keywords: Interactive multimedia, self-modifying protocols, Reconfigurable Petri Nets (RPN), multimedia synchronization, adaptive protocols

1. Introduction

With the recent advances in software and hardware technologies, multimedia systems have become increasingly more sophisticated and complex. In today's world, business demands multimedia applications which combine a variety of sources, such as audio, video, voice, graphics, animation, text and images in real-time, interactive and multiparty communications. On the other hand, the development of Quality of Service (QoS) networks with the ability to guarantee the delivery of time sensitive data have enhanced the transmission of voice, audio and video packets over data networks like local area networks or wide area networks to deliver advanced interactive multimedia services [9]. In order to study and understand such sophisticated system effectively with least effort, modeling is a promising option. It helps to assess vital aspects of a system's performance.

Formalisms such as finite state machines [4, 5, 8] and extended Petri net [2, 3, 7, 23, 25] have been developed to represent temporal information, and support synchronization in runtime rendering for multimedia systems. Others like Wahl and Rothernel's Temporal Model [11], Firefly [11], Fuzzy relation language [11] have also been developed to serve the same purposes. Basically, Petri nets [14] are designed specifically to model systems with concurrent components. Over the years, scientists have extended the Petri net model to overcome the limitation of its original design such that it can be applied to multimedia systems. The focus of the research flows from the synchronization of multimedia without user interactions, to interactions in distributed environments [1-7, 15-16, 21-23]. Some of the models are Object Composite Petri Net, OCPN [15], Dynamic Timed Petri Net [16], Prioritized Petri Net (P-Net) [7], Distributed Object Composite Petri Net: DOCPN [7] and Enhanced Prioritized Petri Net (EP-Net) [23]. The issue of modeling an interactive distributed multimedia application using extended Petri nets is one of the promising areas of research in multimedia synchronization. It has the potential to demonstrate the temporal relationships with interaction modes in distributed environments. The problems of using tradition extended Petri net are the flexibility and

compactness of the model. User interactions (UI) often interrupt current presentation or force a schedule change. Petri net extensions (e.g. Prioritized Petri net, P-net [7], Enhanced Prioritized Petri net, EP-net [23]) have been developed to facilitate UI, however they require sophisticated pre-planning to lay out detailed schedule changes.

On the other hand, there has been active research on self-modifying protocols [20] or adaptive protocols [10, 12, 13] in recent years. A self-modifying (or adaptive) protocol is a set of instructions, rules, or conventions that can be changed by the systems that communicate with the help of that protocol. The purpose of self-modification is to adapt to new communication environment or QoS requirements during protocol execution. Self-modifying protocols do not prevail at present, but this work is certainly important and promising for the next generation of intelligent protocols. Plenty of models developed to model communication protocol execution, to name a few, finite state machines, communicating finite state machines [17, 18, 19], Petri nets. However, there exist no suitable models to simulate protocols that are self-modifying or adaptive during execution.

In this paper, we propose a Reconfigurable Petri Net (RPN) which has multimedia synchronization facilities and the ability to model unpredictable user interactions on the fly. A RPN comprises of a novel mechanism called modifier. This modifier can control, create a new change or delete an existing mechanism (e.g. arc, place, token, transition, etc.) of the net. This important development allows a RPN to have a greater modeling power over other extended Petri nets. The paper includes both the model and theory required to establish the technique's validity and a simulator, which is developed using Visual C++ in Window NT, to show that RPN is feasible.

The modifier empowers RPN to embrace controllability, reconfigurability, and programmability based on Petri net. To model a lip-sync audio and video presentation using OCPN is straightforward as illustrated in figure 1. However, in real life a frame is unrealistic to produce a one-hour movie for example. We might need 108000 frames if it is a 30-frames/sec video playback. Building 108000 x 2 places (i.e. audio and video fragments) in the model is not a pleasant task to do. With the modeling

power from the modifiers, the giant model size is reduced significantly into two layers: control and presentation layers. Moreover, the capability of modifier has enhanced the real-time adaptive modeling power of Petri net.

In this following section, we briefly overview the major extended Petri nets which handle complex multimedia applications. In section 3, we present the definitions and examples of the RPN model. In section 4, we depict synchronous control of user interactions based on RPN. In section 5, we show how RPN can be used to simulate self-modifying protocol execution. Section 6, we explain the RPN simulation, from its architecture to some important functions. Finally, section 7 summarizes the contributions of this research work and potential future directions. The appendix discusses the safeness, boundedness, conservation, and synchronization properties of RPN.

2. Related Work

In this section, we give a broad overview of the existing extended Petri nets which were developed to model interactive distributed multimedia environments. Little and Ghafoor have proposed the use of Object Composition Petri Net, OCPN [15] to model temporal relations between media data in multimedia presentation. OCPN augments the conventional Petri net model with values of time, as duration (d), and resource utilization on the places (p_1, p_2, p_3 and p_4) in the net (figure 1). A token is locked in place p_2 when the duration counter d starts to count down. When the duration reaches zero, the token is considered to be unlocked. The OCPN model has a good expressive power for temporal synchronization. However, it lacks of power to deal with user interactions and distributed environments.

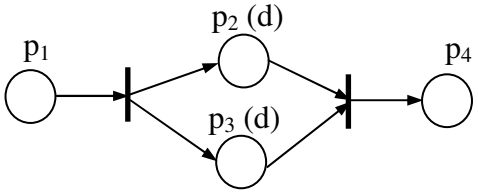


Figure 1. Elements of OCPN

a. The lack of power in OCPN to express user interactions has led to an enhanced OCPN model [16], Dynamic Timed Petri Net (DTPN) proposed by Prabhakaran and Raghavan. DTPN provides the

ability for users to activate operations like *skip*, *reverse*, *freeze*, *restart* and *speed scaling* of presentation. DTPN is suggested to allow a user to pre-empt the Petri net execution sequence and modify the time duration associated with the pre-empted Petri net process. The firing rules are similar to OCPN except with a new feature, escape arcs. An escape arc is ended with a dot instead of an arrowhead. A transition t_j with escape arcs may pre-empt the execution if the other normal input places for t_j are active and contain a locked token and at least one of t_j's escape places becomes nonempty. After pre-emption, a locked token is removed from each of t_j's active input place and a token is added to each of the output place of t_j.

Guan, et al. have proposed DOCPN [7] to overcome the limitation of the original OCPN in modeling interactive distributed multimedia systems. DOCPN inherits the conventional Petri net firing rule, and applies OCPN synchronous methods to synchronize among inter-media objects. Moreover, it extends OCPN to a distributed environment using a global clock, and enables user interaction control into OCPN. Most important of all, a new mechanism known as prioritized Petri net (P-net) is introduced. This priority-input event, as shown in figure 2 [7], has the ability to fire a transition to meet a prescheduled deadline without waiting for the arrival of other non-priority input events. The details of its firing rules are well explained in [7]. The main concern here is what happens if a token arrives at a non-priority input place, after the transition has been forced to fire by an earlier priority input event without the presence of this late arriving token. The paper [7] mentions the processing of late arriving tokens depends on applications; some may choose to discard them (e.g. late arriving video segments), some may choose to recycle or reuse them (e.g. a buffer released for further use). It is better to have a built-in mechanism in P-nets to allow a user/designer to specify how premature/late arriving tokens [23] should be disposed. This motivates the design of EP-nets.

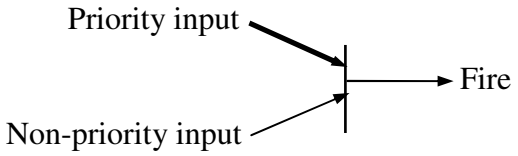


Figure 2. A transition with a priority input [7]

S. U. Guan and S. S. Lim have proposed Enhanced Prioritized Petri Net (EP-net) [23] as an upgraded version of P-net. It has a mechanism known as Premature/Late Arriving Token Handler (PLATH) to handle late and/or premature tokens (locked tokens forced to unlock). Moreover, EP-net imposed another feature that simplifies and improves the flexibility of designing interactive systems, known as dynamic arcs, which can be associated with sets of program statements. For instance, dynamic PLATHs, dynamic input/output events and dynamic priority input events have been developed. Figure 3 shows an example of reverse operation using EP-net. As shown in figure 3, PLATHs (L_1 and L_2) are denoted by the dotted arrows and the dynamic priority input events are signified by the thick 90° arrows with the associated program (ps_1). Even though EP-net has much powerful mechanisms than P-net, having the necessity to duplicate a *reverse* Petri net every time a *reverse* operation is called, implies that the modeling size and effort will be considerably large and difficult to analyze if the model's size increases. Hence, this has induced us to originate RPN.

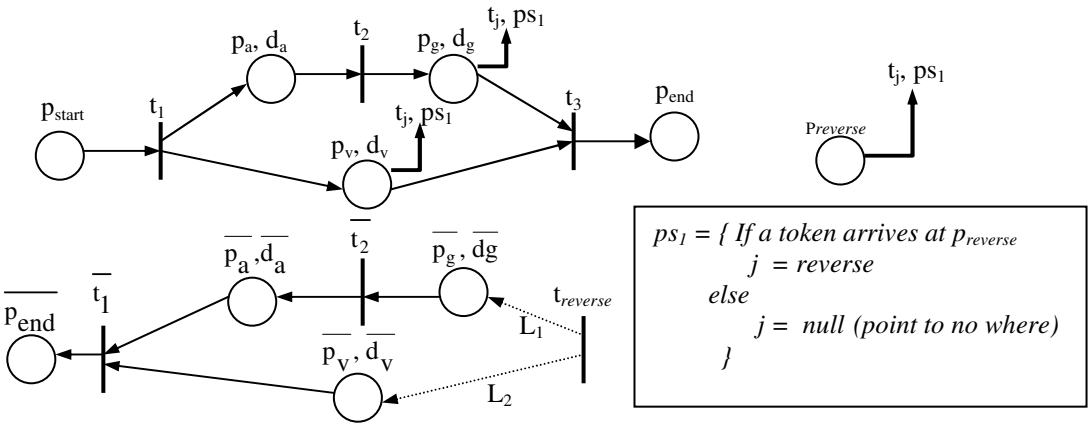


Figure 3. *Reverse* operation using EP-net [23]

SMIL [9] – a W3C standard, allows integrating a set of independent multimedia objects into a synchronized multimedia presentation. Using SMIL, an author can: 1. describe the temporal behavior of the presentation; 2. describe the layout of the presentation on a screen; 3. associate hyperlinks with media objects. The difference between SMIL & RPN is that RPN can deal with the indeterministic part of temporal schedule, i.e. temporal schedule changeable and programmable due to (new type of) user or network interrupts or manipulations at run-

time while SMIL focus on the layout & prescription of temporal schedule due to fixed types of user interrupts. RPN allows run-time change of the temporal model itself with the changes pre-programmed to handle any type of user or network interrupts customizable by the user or network manager.

Last but not the least, a researcher Rüdiger Valk has proposed a Self-modifiable Petri Net (SMPN) on the computation power of extended Petri nets [25]. The paper [25] used the read-write processor to illustrate its SMPN. The reason that we mention this extended Petri net in this section is to clarify that SMPN is different from RPN. As illustrated in the paper [25], a self-modifiable Petri net is defined as in the classic Petri net as a bipartite multi-graph having edges of the forms shown in figure 4.

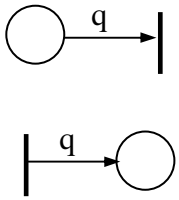


Figure 4. Aspects of Self-Modifiable Petri Net [25]

If q equals to 1, then the firing rule of the transition is defined as in the classic case. On the other hand, q is allowed to be the name of an arbitrary place of the net. In this case the number of tokens to be moved from or to the place is equal to the actual number of tokens specified in q. Consequently self-modifiable nets are able to modify their own firing rules.

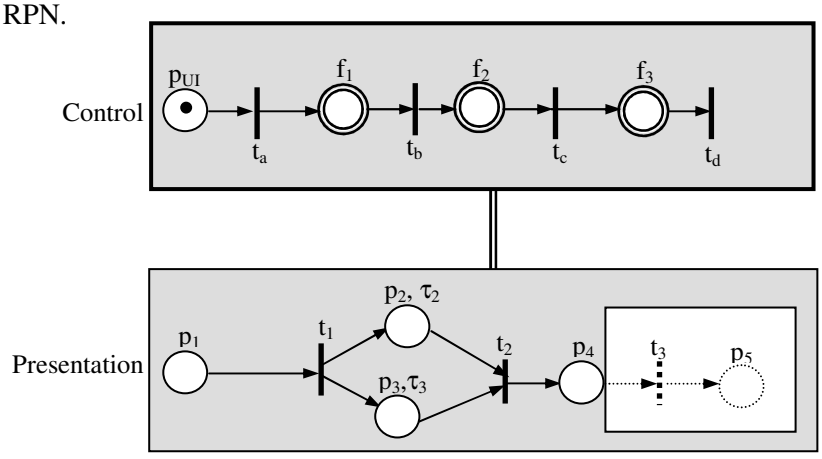
In short, none of the extended Petri nets mentioned above have demonstrated reconfigurability, controllability and programmability as RPN has offered to Petri net. They do not have the ability to reduce the modeling size, improve the design flexibility, model a playback on the fly and simulate a real-time adaptive application with self-modifying operations. OCPN is not able to simulate a distributed environment. XOCPN cannot simulate an interactive environment. DTPN as well as DOCPN, EP-net, and SMPN do not have the ability to reduce its modeling size nor the ability to model a presentation on the fly and simulate real-time adaptive application.

3. Reconfigurable Petri Net

In the following, we introduce the general concepts and definitions of Reconfigurable Petri Net (RPN). RPN methodology facilitates the compact and readable specification of intricate, large-scale synchronization while preserving fine granularity as well as supporting user interactions in distributed environments.

3.1 Definitions

RPN consists of two entities: control and presentation layers. Each entity is represented as a rectangle. These two layers can be joined together by a link (denoted by a double-line arrow). Referring to an example of RPN shown in figure 5, the mechanisms found in the white box displayed in the presentation layer are created after the activation of those modifiers (e.g. f_1, f_2 and f_3) in the control layer. As a matter of fact, there is no white box and mechanisms in it in the beginning. First, the control layer as shown in figure 5 starts with a token in p_{UI}, transition t_a is enabled and fires. The token is removed from p_{UI} and created at modifier f_1. Upon the token arriving at modifier f_1, transition t_3 is then created in the presentation layer. Transition t_b is enabled and fires only if a token is present at f_1 and the transition t_3 is created. After transition t_b is enabled and fires, the token in modifier f_1 is removed and created at modifier f_2. Upon the token arriving at modifier f_2, place p_5 is created in the presentation layer. Next, transition t_c is enabled and fires. The token in modifier f_2 is removed and created at modifier f_3. Upon the token arriving at modifier f_3, the arcs p_4t_3 and t_3p_5 are created in the presentation layer. Finally, transition t_d is enabled and fires. The token is removed from modifier f_3. Therefore, we have shown how a RPN works. In the following, we explain the definitions related to RPN.



Legends:

$COM(f_1)$ = {add transition t_3}

$COM(f_2)$ = {add place p_5}

$COM(f_3)$ = {add arcs p_4t_3, t_3p_5}

Figure 5. RPN: an example

Definition 1: Control and Presentation Layers

The structure of an unmarked layer in RPN is a six-tuple, S = {T, P, A, D, L, COM}. For a marked RPN layer, the definition of the structure becomes a seven-tuple, S = {T, P, A, D, L, COM, M}. Referring to the structures S as mentioned above, the condition $P \cap T = \varnothing$ holds. A complete RPN may consist of zero or more control layers and one or more presentation layers.

$T = \{t_1, t_2, t_3, \ldots\ldots\ldots\ldots, t_m\}$ is a finite set of transitions where $m > 0$.

$P = \{p_1, p_2, p_3, \ldots\ldots\ldots, p_i, f_4, f_5, f_6, \ldots\ldots\ldots..f_k\}$ is a finite set of places and/or modifiers where i and $k > 0$.

$COM: f_a \rightarrow \{com_1, com_2, com_3 \ldots\ldots com_z\}$ is a mapping from the set of modifiers to the commands (as defined in Table 1) where a and $z > 0$.

$A: \{P \times T\} \cup \{T \times P\}$ is a set of arcs representing the flow relation.

$M: P \rightarrow I^+$, $I^+ = \{0, 1, 2\ldots\}$ is a mapping from the set of places or modifiers to the integer numbers, representing a marking of a net.

$D: p_b \rightarrow R^+$ is a mapping from the set of places to the non-negative real numbers, representing the presentation intervals or the durations for the resources concerned where $b > 0$.

$L = \{c_x$ or $p_x\}$ indicates whether an entity is a control layer c_x or presentation layer p_x where $x > 0$.

We use a conventional set of graphical symbols for nets in which places are represented as circles, transitions represented as a solid bar and arcs represented as a unidirectional arrow. In addition, a modifier (f) is represented as a double, control layers are represented as rectangles with bold borderline and presentation layers are represented as rectangles with fine borderline.

No.	Mechanisms	Commands	Actions
1.	Arc	Disable arc	An arc is disabled (Virtually deleted).
		Enable arc	An arc is enabled (Recovered).
		Create arc	An arc is created.
		Delete arc	An arc is deleted.
		Reverse arc	The direction of an arc is *reversed*.
2.	Place or Modifier	Create place or modifier	A place or modifier is created.
		Delete place or modifier	A place or modifier is deleted.
		Replace place or modifier	A place or modifier is replaced.
3.	Transition	Disable transition	A transition is disabled (Virtually deleted).
		Enable transition	A transition is enabled (Recovered).
		Create transition	A transition is created.
		Delete transition	A transition is deleted.
4.	Token	Lock token	To lock a token (The duration continues to count down, however when the count reaches zero, the token remains lock).
		Unlock token	To unlock a token (The duration forces to zero and the token is unlocked).
		Pause token	To stop counting down if a place associated with a duration or stops a transition to be fired if the place is associated with no duration.
		Resume token	To resume a token and start counting from the time it has been paused.
		Create token	To create a token to the indicated place with no condition.
		Delete token	To remove a token at the indicated place with no condition.

Table 1. List of commands

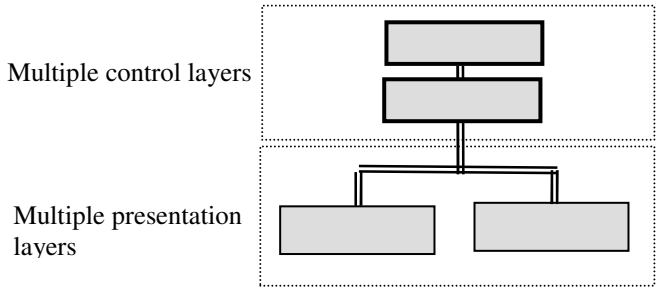


Figure 6. Illustration of multiple control and presentation layers

In general, a control layer may consist mainly of modifiers and a presentation layer may consist mainly of places representing resource playback. The layer names have self-explained their meanings. A presentation layer represents a layer that consists of multimedia resources playback. Otherwise, the layer is known as control layer. However, there is a special case see *definition 3* for more details. We can also extend a single control and presentation layer into multiple control and presentation layers as shown in figure 6. This multiple control and presentation layers operate on a hierarchy concept. The control layers is vertically integrated and a single piece of integrated control layers is applied to numerous presentation layer concurrently.

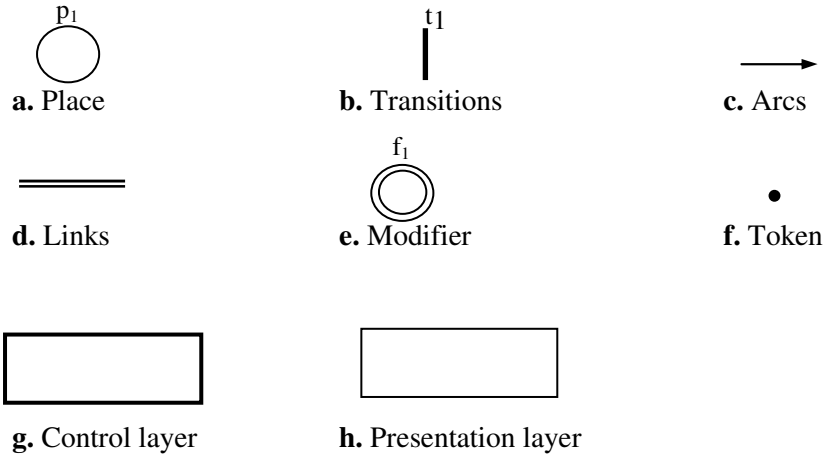


Figure 7. Graphic representations

The set of graphical symbols for RPNs are demonstrated in figures 7a to 7h. A classic place is shown in figure 7a, which could represent a resource (e.g. audio, video playback, etc). If the place is associated with duration (D), it indicates the interval of the resource to be consumed. Figure 7b displays a transition, which represents a synchronization point in a presentation. In figure 7c, an arc is demonstrated which represents a flowing relation in a presentation. Then, the links as shown in figure 7d establish connections linking two different layers (e.g. between control and presentation/control layers). Figure 7e introduces a modifier, which signifies a place having the ability to control, create or delete a new or existing mechanism (e.g. arc, place, token and transition) of a presentation/control layer in a net. A solid dot (token) as displayed in figure 7f indicates the marking in a place. Finally,

the two rectangles denote a control and a presentation layer as presented in figures 7g and 7h respectively.

***Definition 2*:** Firing rules

The conventional firing rules of a Petri net are as the following. A transition is enabled when all its input places comprise of number of tokens greater than or equal to the number of each respectively place's arcs to the transition. If the condition mentioned is met, the transition fires and token(s) is removed from each of its input places and token(s) is created at each of its output places. The transition fires instantly if each of its input places is not associated with any duration or contains any unlocked token. In case when a place is associated with duration, the place remains in the active state for an interval specified by the duration d_1 after receiving a token. During this period, the token is locked. Upon the cessation of the duration d_1, the token becomes unlocked.

RPN extends the capabilities of OCPN by providing support for interactive, adaptive multimedia environment and enhance the modeling power over the latest extended Petri net (e.g. P-net, EP-net, etc). This is achieved by using some novel mechanisms: modifiers. The two entities of a RPN namely: control and presentation layers are designed to distinguish between layers for ease of analysis. With mechanisms grouped into layers, modifiers can modify them in terms of group instead of individual. In a way, this helps reducing the modeling size. However, the grouping approach is not the key factor to reduce the modeling size. The key factor is the power introduced by the programming modifiers as illustrated in section 4 & 5. Once the mechanisms are grouped into layers, how the layers are communicated is indicated by links.

Unlike the conventional places and transitions, modifiers f have the capabilities to manipulate or customize the presentation flow and the schedule of multimedia in a presentation layer(s) or controlling flow and schedule in the other control layer(s). A modifier f executes its associated commands (see table 1) upon receiving a token. In addition to the conventional firing rules mentioned above, if a transition has an input modifier containing a token, the transition is enabled and fires upon the modifier's command being executed. These are also known as inter-layer firing rule.

Inter-layer firing rules are applied at layers level. Control and presentation layers interact whenever execution of modifier-associated commands (see table 1) in a layer(s) manipulates and changes mechanisms in another layer(s). We have two types of interactions between the layers. They are feed-forward and feedback interactions. The former interaction occurs when modifiers f in a layer change the structure of another layer (e.g. to change the course of presentation style, flow, etc). The latter interaction occurs when a layer reports the state of it to another layer. For an example, we can use the virtual common transition driving two output places (one in a presentation layer and the other in a control layer) as shown in figure 9. Regardless the interactions are feed-forward and/or feedback between two layers, we need to consider the synchronization issue among layers. For example in figure 10, modifier f_3's command is to disable transitions t_1 and t_2. At this moment, there are tokens in modifier f_3 and place p_1 which means that the command can execute (transition t_a is enabled) and transition t_2 can fire. Therefore, we have a racing problem. We have two situations: First, transition t_2 fires before the command is executed (not desired). Second, the command executes before transition t_2 fires (desired). In order to overcome the problem, we have two rules of thumb: First, the execution of a modifier's command has higher priority than the firing of a. Second, the lock-step synchronization or interleaved approach is applied to the control and presentation layers. Enabled transitions within a layer (i.e. control layer) will fire before enabled transitions in the other layer (i.e. presentation layer). Starting with the control layer, each layer takes its turn to execute its respective mechanisms until no enabled transitions can fire.

Definition 3: Self Reconfigurable Petri net (S-RPN)

S-RPN is a sub-class of RPN. S-RPN does not have separate control and presentation layers. The modifiers are able to control, create, or delete new or existing mechanisms in the same layer (see section 3.2, *theorem 2*).

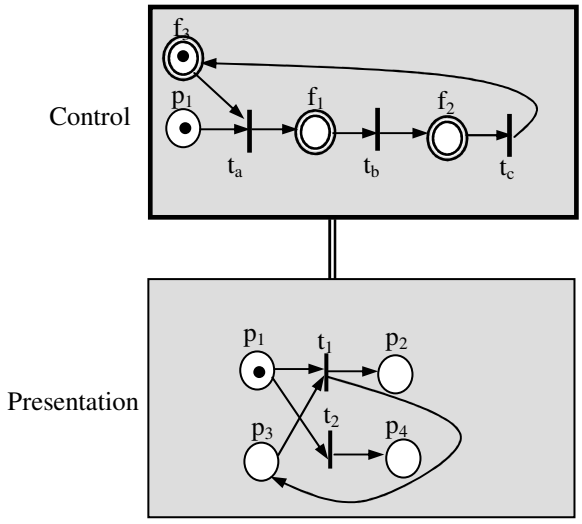
3.2 Modeling Power of RPN and S-RPN

In this section, we present examples to elaborate upon the concepts discussed earlier, and describe how the Turing machine can be simulated from RPN or S-RPN. Agerwala et al. [24] have

shown that an extended Petri net model with the ability to test a place for zero token can simulate a Turing machine. Thus by proving a RPN/S-RPN has the ability to test a place for zero token it shows that they have the modeling power of Turing machines.

Theorem 1: RPN has the modeling power of Turing machines.

Consider p_3 represents a place to be tested, p_1 represents a place to start the zeroing test, p_4 represents p_3 has no token and p_2 represents p_3 has a token as shown in figure 8. Note that p_1 (see figure 8) appears in the control and presentation layers. To illustrate how it happens, we can imagine that they are driven by a common transition as shown in figure 9.



Legends:

$COM(f_1)$ = {enable transition t_1}

$COM(f_2)$ = {enable transition t_2}

$COM(f_3)$ = {disable transitions t_1 and t_2}

p_1 represents a place to start the zeroing test.

p_2 represents p_3 has a token.

p_3 represents the place to be tested.

p_4 represents p_3 has no token.

Figure 8. Simulated zero-token test by RPN

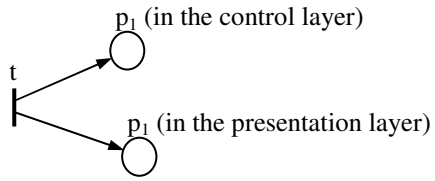
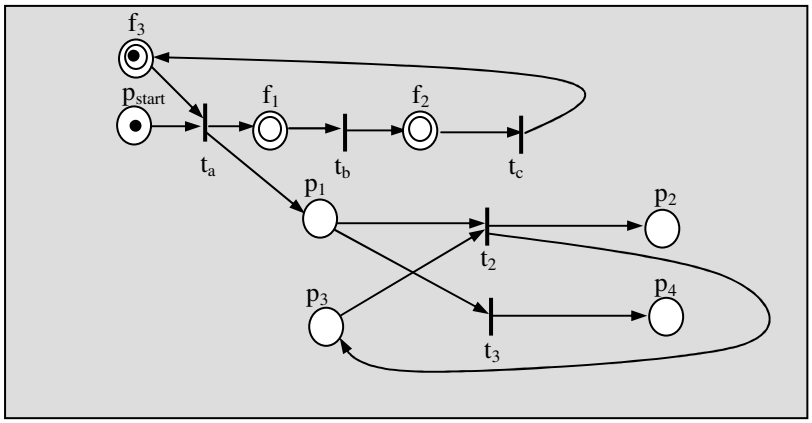


Figure 9. Driven by a virtual common transition (t)

Now, we are ready to explain how figure 8 works. The initial marking is marked as shown in figure 8. The zeroing test starts off with tokens arrive at p_1s as shown in figure 8. Assume p_3 (in the presentation layer) contains no token. Initially, the transitions t_1 and t_2 are disabled by modifier f_3. Transition t_a (in the control layer) is enabled and fires. Tokens in place p_1 and modifier f_3 (in the control layer) are removed and the token is created at modifier f_1. Upon the arrival of token at modifier f_1, transition t_1 (in the presentation layer) is enabled. Therefore, the arrival of token in f_1 enables t_1 but t_1 is not able to fire (no token in p_3). Transition t_b fires and the token is removed from modifier f_1 and created at modifier f_2. So, transition t_2 is enabled by modifier f_2 and together with a token in place p_1, transition t_2 fires. As a result, p_2 has no token and p_4 contains a token which indicates that p_3 contains no token. Repeat the whole procedure again but this time p_3 contains a token. The outcome is p_2 contains a token and p_4 contains no token. Therefore, we have established a proof that RPN has the modeling power of Turing machines. The purpose of the output arc t_1p_3 as shown in figure 8 is to preserve the token in p_3 if t_1 fires.

Theorem 2: S-RPN has the modeling power of Turing machines

Unlike RPN, S-RPN has the power to change the firing rule within its layer. Let us illustrate it as shown in figure 10.



Legends:

$COM(f_1)$ = {enable input arcs p_1t_2 and p_3t_2} $COM(f_2)$ = {enable input arc p_1t_3 }

$COM(f_3)$ = {disable input arcs p_1t_2, p_3t_2 and p_1t_3}

p_{start} represents the start of zeroing test.

p_2 represents p_3 containing a token.

p_3 represents the place to be tested.

p_4 represents p_3 containing no token.

Figure 10. Simulated zero-token test by S-RPN

The initial marking is marked as shown in figure 10. Initially, the input arcs p_1t_2, p_3t_2 and p_1t_3 are disabled by modifier f_3. Transition t_a is enabled and fires. The tokens in place p_{start} and modifier f_3 are removed and created at modifier f_1. Upon the arrival of token at modifier f_1, the input arcs p_1t_2 and $p_3t_2$are enabled. Assume initially, place p_3 contains no token, transition t_2 is not enabled and hence cannot fire. Then, transition t_b is fired and the token in modifier f_1 is removed and creates at modifier f_2. Upon the arrival of a token at modifier f_2, the input arc p_1t_3 is enabled. Hence, transition t_3 is enabled and fires. As a result place p_2 contains no token and place p_4 contains a token. The procedure is repeated with a token in place p_3. This time, the consequences show that place p_2 contains a token and place p_4 contains no token. We have proved that S-RPN is as powerful as a Turing machine. From this example and the example shown in figure 10, they have shown the flexibility of using either arcs or transitions enabling techniques to prove that they are as powerful as a Turing machine.

3.3 Synchronization

Extended Petri nets are so popular in modeling multimedia presentation because they exhibit the synchronization properties among resources e.g. lip-sync. In this section, the term synchronization refers to the synchronization between layers. In order to prevent any conflict between the layers, a control layer should pause the token(s) in the presentation before carrying out its necessary executions as shown in figures 11, 12 and 13 for examples. In case of controlling the presentation on fly, the user needs to anticipate outcome of his design to avoid any adverse result. For example, it is obvious that

we cannot create and delete the same mechanism (e.g. token, place, transition, arc, etc) at the same time. The effect is undetermined. Take figure 8 for an example, upon the arrival of tokens at places p_1s and modifier f_3, transitions t_a and t_1 are enabled and fire only after modifier f_3 executes its command. The command is to disable the transitions t_1 and t_2, if transition t_1 fires before the command is executed in the modifier f_3 then the outcome is not desired. However, if the rules of thumb as stated in section 3.1 *definition 2* are followed, the racing problem is avoided.

4. Synchronous Control of User Interactions

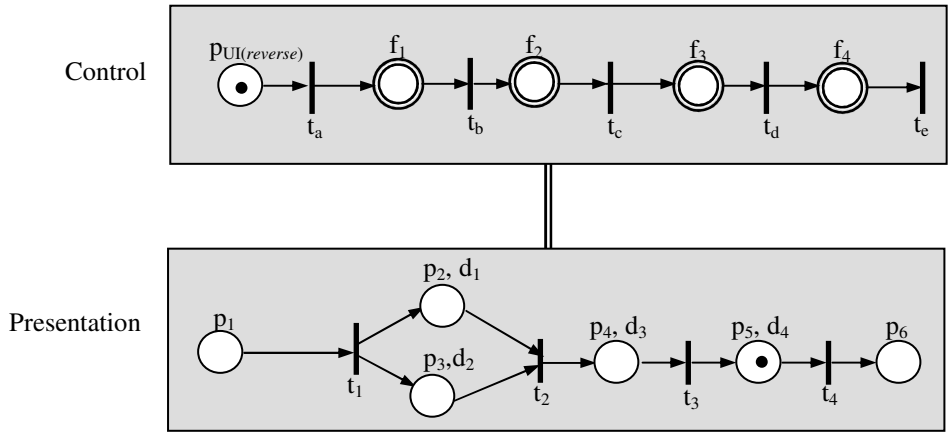
The synchronization mechanism (in the control layer): a modifier has the authority to manipulate the existing mechanisms or generate new mechanisms (in presentation layers). This enhances the power to support user interaction. Whenever a user interacts, a token arrives at the initial place p_{UI} as shown in sections 4.1-4.4. As the token flows through the RPN structure in the control layer, the modifiers with each associated commands are executed respectively and the interactions are carried out properly. In addition, we have included an example in section 4.5, demonstrating how RPN has the ability of reducing modeling size as compared to other extended Petri nets

4.1 *Reverse* Operation

The *reverse* operation is similar to the *forward* operation, only that the presentation flows are opposite. Sometime, the *reverse* operation can also be combined with the *speed scaling* operation to form a *fast reverse* operation. To simplify the explanation, let us illustrate the *reverse* operation without *speed scaling* as shown in figure 11.

When a user requests for a *reverse* operation, a token is created at $p_{UI(reverse)}$. t_a (in the control layer) is enabled and fired. It removes the token in $p_{UI(reverse)}$ and creates a token in f_1. The arrival of the token in f_1 causes its associated command to be executed: pause the token in the presentation layer. Then, the token in the control layer flows through f_2, f_3 and f_4 consequently. When a token arrives at f_2, it changes the direction (*reverse*) of all the arcs in the presentation layer. Next, when a token arrives at f_3, it replaces the forward resources (places) with the reverse ones. Last, when a token arrives at f_4, the token in the presentation layer is resumed. The *reverse* operation has been executed

and the presentation flows in the *reverse* manner. The places p_2 to p_5 shown in figure 11 are illustrated in coarse-grain. A reverse resource implies that a resource with its intra-media frames are arranged in a *reverse* manner such as a reverse audio playback. For a static media (e.g. image, text, etc), we see no *reverse* effect during its interval (e.g. d_3 and d_4) if a user requests for a *reverse* at that instant. The viewer needs to watch the intact presentation to realize that the *reverse* is in operation.



Legends:

COM(f_1) = {pause the token in the presentation layer}

COM(f_2) = {*reverse* all arcs in the presentation layer}

COM(f_3) = {replace all places with the resources *reversed* in the presentation layer}

COM(f_4) = {resume the token in the presentation layer}

p_1 represents the start of a multimedia presentation.

p_2 represents an audio playback with duration d_1.

p_3 represents a video playback with duration d_2.

p_4 represents an image display with duration d_3.

p_5 represents a text display with duration d_4.

p_6 represents the end of a multimedia presentation.

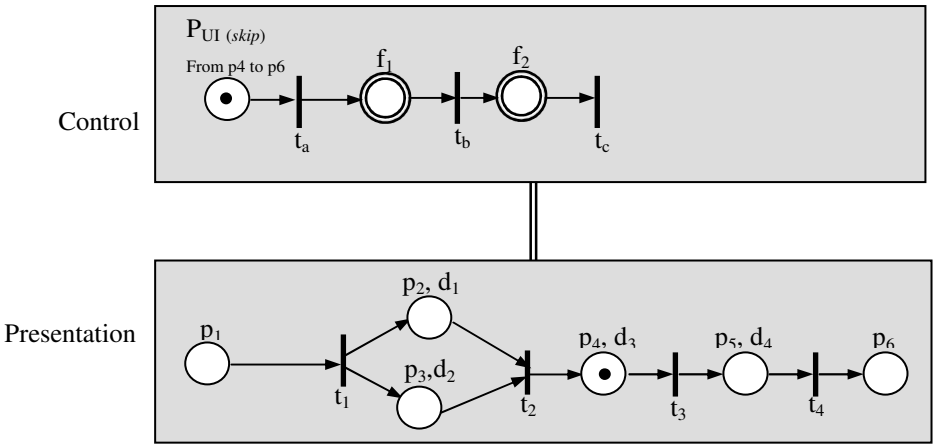
Note that p_1 to p_6 share the same representatives throughout section 4.

Figure 11. *Reverse* operation

4.2 *Skip* Operation

A user might feel that a certain section of a presentation boring and decides to skip. This operation is able to *skip* on the fly an on-going stage or a stage which is going to be presented as specified by the user. Assume a multimedia presentation is displaying an image (p_4, in the presentation layer) when a user decides to *skip* from the current stage to the end of the presentation (t_4, in the presentation layer) as demonstrated in figure 12.

Upon a user interaction, a token arrives at $p_{UI(skip)}$. The token is created and moves from f_1 to f_2 (in the control layer) as the transitions t_a to t_e are enabled and fire respectively. When the token arrives at f_1, it executes the command to delete the token in presentation layer. Subsequent, when a token arrives at f_2, a token is created at place p_6 and the presentation comes to an end.



Legends:

COM(f_1) = {delete the token in the presentation layer}

COM(f_2) = {create a token at place p_6}

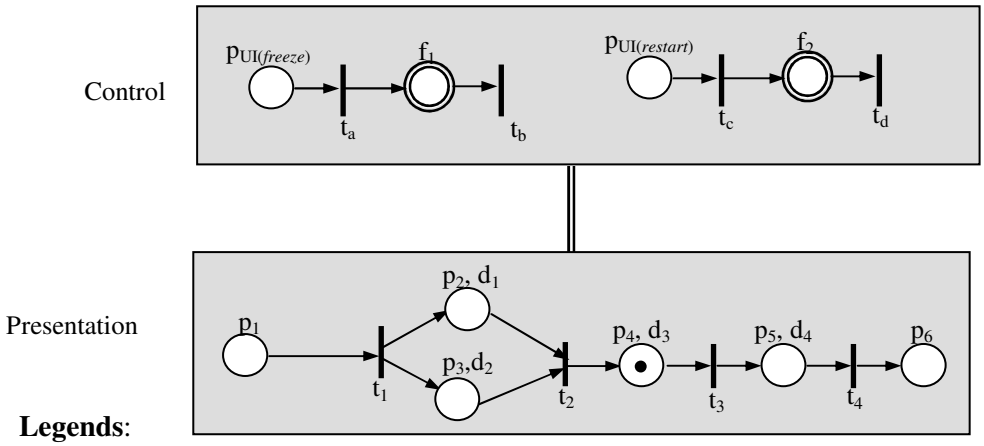
Figure 12. *Skip* operation from p_4 to p_6

4.3 *Freeze* and *Restart* Operations

Among various user interactions, *freeze* and *restart* operations are the simplest ones to model. The RPN model for *freeze* and *restart* operations is shown in figure 13.

When a user requests for a *freeze* operation, a token is created in $p_{UI(freeze)}$ (in the control layer). t_a is enabled and fires. Upon t_a firing, a token is removed from $p_{UI(freeze)}$ and a token is created in f_1. The arrival of token at f_1 would run the command to pause the token in the presentation layer. In other

words, the presentation is paused. Then, when the user requests for a *restart* operation, a token is created in $p_{UI(restart)}$ which enables and fires t_c. With that, the token in the presentation layer is resumed and the presentation starts to playback.



Legends:

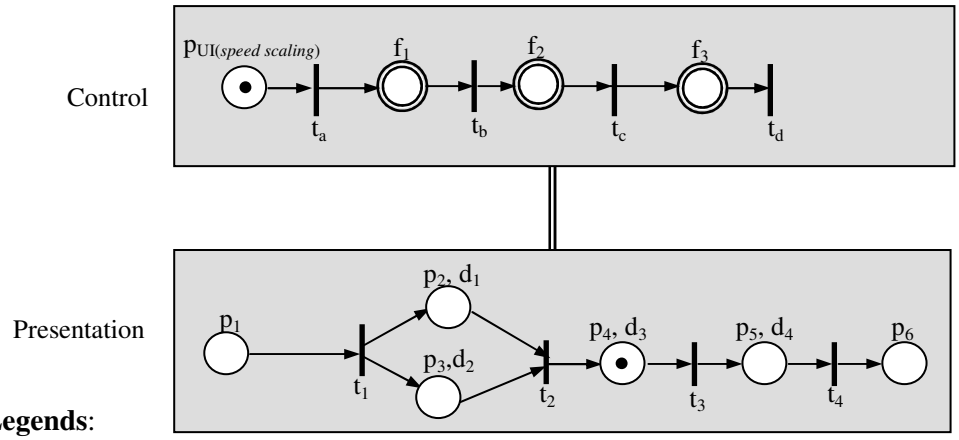
COM(f_1) = {pause the token in the presentation layer}

COM(f_2) = {resume the token in the presentation layer}

Figure 13. *Freeze* and *restart* operations

4.4 Speed Scaling and Zooming Operations

A user can either increase or decrease the speed of a presentation by a factor of 2, 3, etc. in the forward or reverse manner. We have demonstrated a fast forward example as shown in figure 14.



Legends:

COM(f_1) = {pause the token in the presentation layer}

COM(f_2) = {replace all places with resources of presentation speed increment by 2}

COM(f_3) = {resume the token in the presentation layer}

Figure 14. *Speed scaling* operation

Whenever a *speed scaling* operation is requested, a token is created in $p_{UI(speed\ scaling)}$. t_a is enabled, fires and the same procedure happens as mentioned in sections 4.1, 4.2 and 4.3, only that f_1, f_2 and f_3 carry different execution in the presentation layer. A token arriving at f_1 pauses the token in the presentation layer. The places are replaced with resources of presentation speed incremented by a factor of 2 upon a token arriving at f_2. Last, when a token is created in f_3, the presentation is resumed with the new presenting speed. Considering a case when a user requests to increase the speed of the presentation by a factor of 2, it means that the playback skips every odd/even sequence of its intra-media. On the other hand, when a user requests to decrease the speed of the presentation by a factor of 2, it connotes that the playback includes a duplicated intra-media frame for every odd/even frame.

Similar to speed scaling, zooming in or out of a video screen on the fly during a video conference at t_1 and t_2 as shown in figure 14 can also be simulated. The modifier command (f_2) will be changed as to replace all video places with larger or smaller video screens.

4.5 RPN has the Ability of Reducing Modeling size as Compared to Other Extended Petri Nets

To model a lip-sync presentation of a series of video frames and audio samples for example 1000 frames or samples Base on existing extended Petri nets, the user needs to create about 2000 places (representing the video frames and audio samples) as shown in figure 15a. As a result, this has become an intricate and time-wasting task for the user.

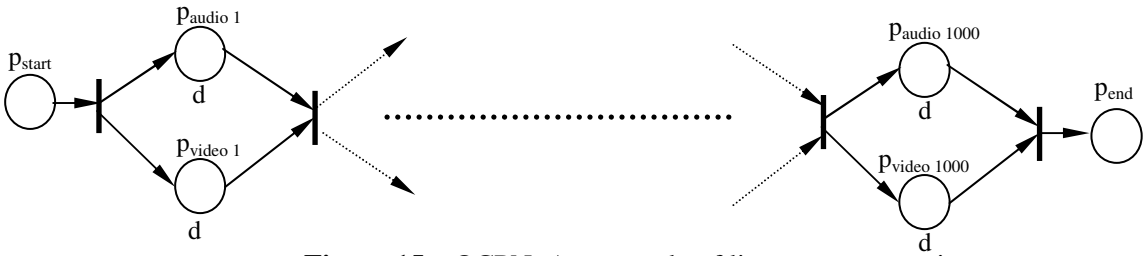
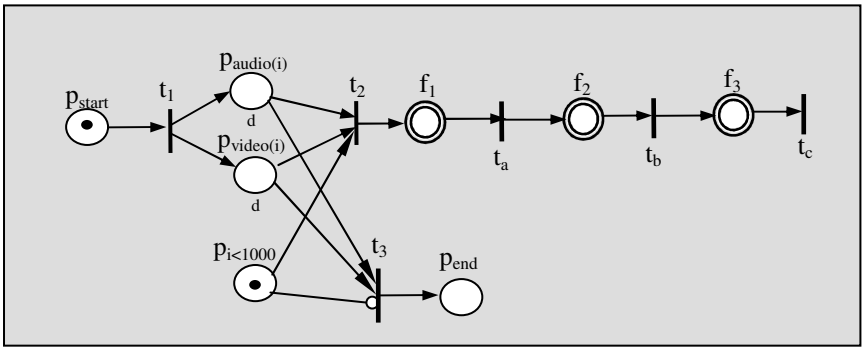


Figure 15a. OCPN: An example of lip-sync presentation

We demonstrate how RPN can reduce the modeling size in figure 15b. As we can see RPN uses only 8 places and modifiers to represent the 1000 video frames and 1000 audio samples. Even if the number of frames and samples increase, the number of places in the RPN as shown in figure 15b

remains. This is how figure 15b works. Initially, the counter's value i is set to 1. When a token is created at place p_{start}, transition t_1 is enabled and fires. Token in place p_{start} is removed and create at places $p_{audio(1)}$ and $p_{video(1)}$. Since place $p_{i<1000}$ always contains a token if the counter value is less than 1000. Upon the tokens in places $p_{audio(1)}$ and $p_{video(1)}$ are unlocked, transition t_2 is enabled and fires. As the token flow through modifiers f_1, f_2 and f_3, places $p_{audio(1)}$ and $p_{video(1)}$ is replaced by places $p_{audio(2)}$ and $p_{video(2)}$ respectively with each places having a token. The procedure repeats itself until when the counter's value i is 1000. At this moment, place $p_{i<1000}$ contains no token. Upon the tokens in places $p_{audio(1000)}$ and $p_{video(1000)}$ are unlocked, transitions t_3 instead of t_2 is enabled and fires. The reason is that place $p_{i<1000}$ has an inhibitor (i.e. a mechanism of classic Petri net) pointing to transition t_3. This input event (a line with a circle at its end) is activated when its place contains no token. Finally, tokens in places $p_{audio(1000)}$ and $p_{video(1000)}$ are deleted and created at place p_{end}.

Presentation



Legends:

$COM(f_1)$ = {delete places ($p_{audio(i)}$ and $p_{video(i)}$), create places ($p_{audio(i=i+1)}$ and $p_{video(i=i+1)}$)}

$COM(f_2)$ = create input arcs ($p_{audio(i=i+1)}t_2$, $p_{video(i=i+1)}t_2$, $p_{audio(i=i+1)}t_3$, $p_{video(i=i+1)}t_3$).

$COM(f_3)$ = create tokens at places ($p_{audio(i=i+1)}$ and $p_{video(i=i+1)}$).

p_{start} represents the start of presentation.

$p_{audio(i)}$ represents the audio playback.

$p_{video(i)}$ represents the video playback.

$p_{i<1000}$ represents the index count less than 1000.

i represents the sequence number of frame and sample.

d represents the duration of the resource (i.e. each video frame and audio sample) playback.

Figure 15b. RPN: An example of lip-sync presentation

5. Using RPN to Model Self-Modifying ABP

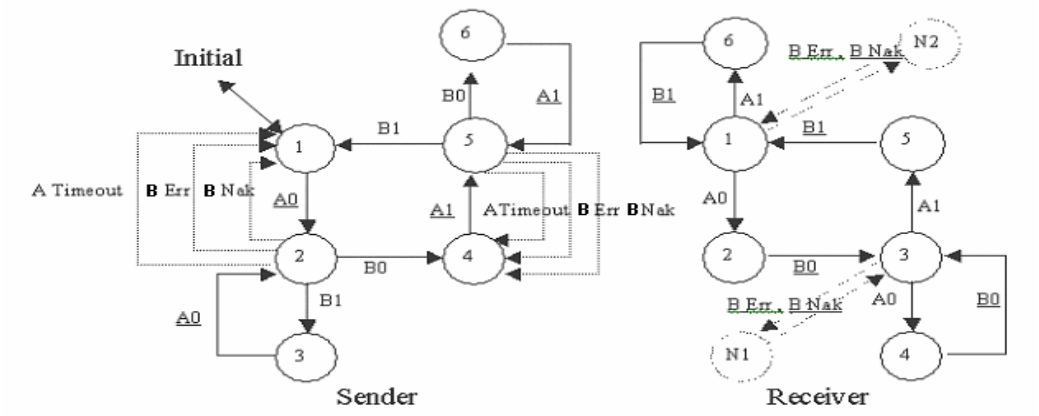
In the following we show how RPN can be used to model self-modifying protocol execution. First we give a self-modifying protocol example, and then we show how RPN can be used to model its execution.

Self-Modifying Protocol (SMP) is a set of instructions, rules, or conventions that can be changed by the systems that communicate with the help of that protocol.

5. 1 Self-Modifying Alternating Bit Protocol

The original Alternating Bit Protocol (ABP) can be defined as follows:

- The sender sends its data messages, one by one, to the receiver, but after sending each data message, it must wait for an acknowledgement before sending the next data message.
- Whenever the receiver receives a data message, it should be able to detect whether it has received an identical copy of this message earlier. For this reason, the value of some bit in the sender is attached to each data message sent. So long as a data message is being resent, the value of this bit remains fixed, but whenever a new data message is about to be sent, the value of this bit is altered (hence the name "alternating bit").



Legends:
Solid line: original ABP
Dotted line: self-modifying part

Figure 16. Extension of the Alternating Bit Protocol

ABP is designed for communication without error, which is the common scenario during data transmission. Occasionally, network congestion or error can occur, then ABP is not sufficient. SMP (e.g. Self-Modifying ABP) has been proposed in [20] so that it is lightweight during normal communication, however it is capable of self-modification to deal with exceptions when network events arise. An SMP example has been presented in [20] as shown in figure 16. When the sender does not receive any acknowledgement for some time or receives a negative acknowledgment (Nak) / corrupted message (Err), some new transitions will be added to the sender and some new states (N_1, N_2) and transitions will be added to the receiver. Self-modification is introduced upon serious network events. The protocol change created is meant to deal with the event raised.

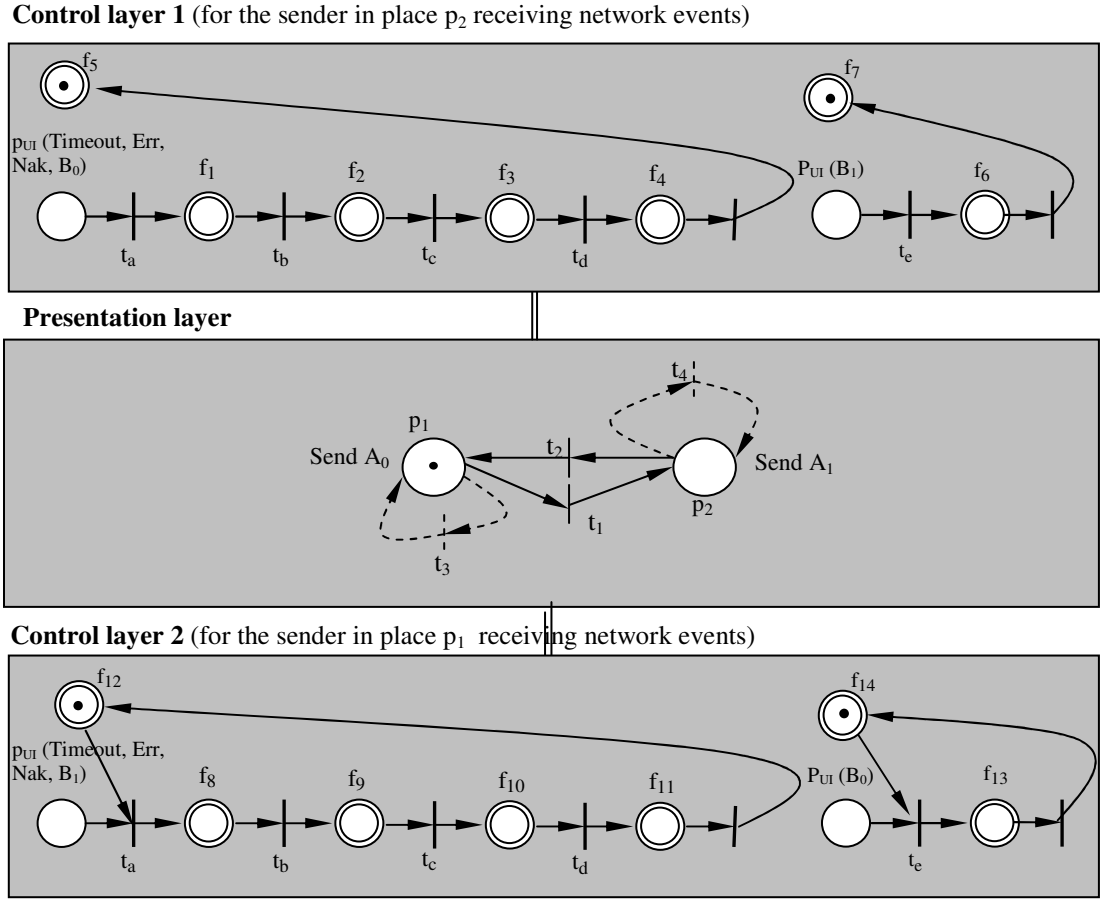
5.2 Self-Modifying ABP Modeling using RPN

Using RPN to model ABP self-modification, an incoming event (e.g. message received, message error or loss) will trigger an interrupt upon which a corresponding color token is created based on the type of events occurred or messages received. Then transitions in the corresponding control layer will fire, the commands associated with modifiers will be executed upon the arrival of a token. Different events/messages received will lead to different implementation as shown in figures 17 & 18. We'll give a detailed description in the following.

1) Receiver's responses in place p_1:

In figure 18, if the receiver (place p_1) receives message A_0, that means a message is received, a token corresponding to this will be injected into P_{UI} (A_0) in the control layer 2, the commands associated with it in this control layer will be executed. Transition t_1 in the presentation layer will be enabled. Then B_0 is sent to the sender and t_1 fires – while the resource token moves from p_1 to p_2, the next stage begins. Otherwise, if a Nak or Err signal is detected/received, then the receiver is alerted that there is some problem with communication. A token used to handle this condition will be injected into p_{UI} (Err, Nak), the commands associated with it in the control layer 2 will be executed, a new transition t_3, a new place p_3 and four arcs will be created represented by the dot lines as shown in

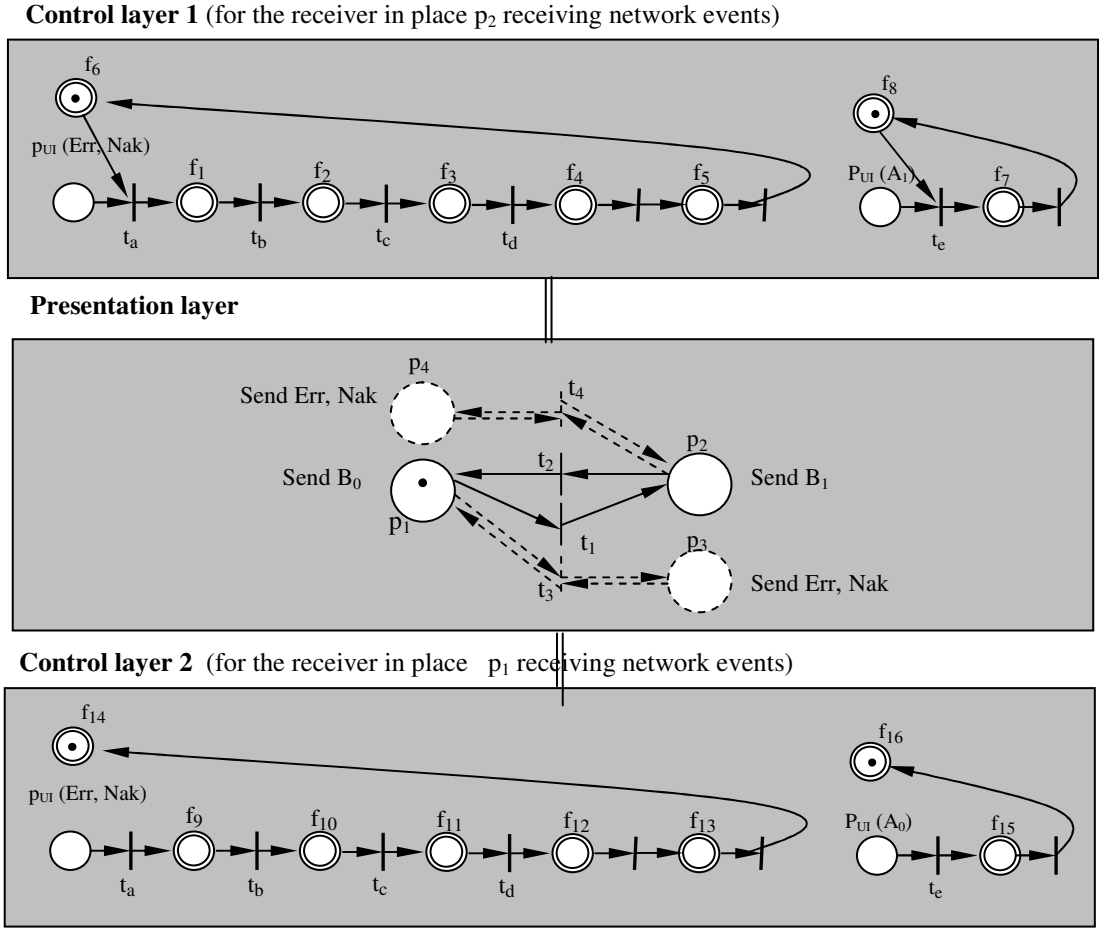
figure 18. Then transition t_3 fires, the resource token moves to p_3 and Nak/Err is sent to the sender. At last, the resource moves back to p_1 - waiting for the message to be resent from the sender.



Legends:

place	Events (token injected into p_{UI})	Actions (semantics)
p_1 upon receiving messages/events	Timeout, Err, Nak, B_1	COM(f_8) = {create transition t_3} COM(f_9) = {create arc p_1t_3} COM(f_{10}) = {create arc t_3p_1} COM(f_{11}) = {enable transition t_3} COM(f_{12}) = {disable all transitions}
	B_0	COM(f_{13}) = {enable transition t_1} COM(f_{14}) = {disable all transition}
p_2 upon receiving messages/events	Timeout, Err, Nak, B_0	COM(f_1) = {create transition t_4} COM(f_2) = {create arc p_2t_4} COM(f_3) = {create arc t_4p_2} COM(f_4) = {enable transition t_4} COM(f_5) = {disable all transitions}
	B_1	COM(f_6) = {enable transition t_2} COM(f_7) = {disable all transitions}

Figure 17. Using RPN to design Self-Modifying ABP – Sender



Legends:

place	Events (token injected into p_{UI})	Actions (Semantics)
p_1 upon receiving messages/events	A_0	COM(f_{15}) = {enable transition t_1} COM(f_{16}) = {disable all transitions}
	Message loss (Nak) or message error (Err)	COM(f_9) = {create transition t_3} COM(f_{10}) = {create place p_3} COM(f_{11}) = {create arc p_1t_1 & t_1p_3} COM(f_{12}) = {create arc p_3t_3 & t_3p_1} COM(f_{13}) = {enable transition t_3} COM(f_{14}) = {disable all transitions}
p_2 upon receiving messages/events	A_1	COM(f_7) = {enable transition t_2} COM(f_8) = {disable all transitions}
	Message loss (Nak) or message error (Err)	COM(f_1) = {create transition t_4} COM(f_2) = {create place p_4} COM(f_3) = {create arc p_2t_4 & t_4p_4} COM(f_4) = {create arc p_4t_4 & t_4p_2} COM(f_5) = {enable transition t_4} COM(f_6) = {disable all transitions}

Figure 18. Using RPN to design Self-Modifying ABP - Receiver

2) Sender's responses in place p_1:

As shown in figure 17, after the sender (place p_1) sent a message A_0 to the receiver, if a timeout, Err, Nak or B_1 signal is detected or received, then the sender is alerted that transmission has not been successful. Then a token used to handle this condition will be injected into p_{UI} (Timeout, Err, Nak, B_1), the commands associated with it in the control layer 2 will be executed. A transition t_3 and two arcs p_1t_3 & p_3t_1 are created in the presentation layer, represented by the dot lines as shown in figure 17. At last, transition t_3 in the presentation layer fires, a resource token is re-injected into place p_1, the current message A_0 will be sent again. If B_0 is received, a token corresponding to this will be injected into P_{UI} (B_0) in the control layer 2, the commands associated with it in this control layer will be executed. Transition t_1 in the presentation layer will be enabled. At last, transition t_1 fires - while the resource token moves from p_1 to p_2, the next message A_1 is sent.

6. RPN Simulation

The modifiers and layers (i.e. control and presentation layers) of RPN are introduced to enable programmers to specify user interactions. The presentation specification mechanisms like places, input/output events and transitions are developed to enable programmers to specify temporal relationships among media in a presentation. Together, the distributed interactive multimedia applications can be simulated using this RPN simulator. These mechanisms might be grouped into many different presentation layers whereby some of these layers might be monitored and manipulated by other control layers. This is an interesting issue because we have formed an object-oriented approach to the model. The control layer uses to control a presentation layer, can also use to control other layers in future. With these two powerful mechanisms: modifiers and layers, the modeling size is reduced significantly as illustrated in section 4.

RPN simulator is designed to be user-friendly. What a programmer needs is a mouse that does most of the job. To draw a place, modifier, or transition, the user just clicks on the place, modifier or transition icon shown on top of the menu as displayed in figure 19, and keys in an integer label from 1

to 50. In the current prototype, we have set its maximum label to 50. Then, by clicking onto any area within the white screen (see figure 19), a place, modifier, or transition will be drawn. With that, the places or modifiers and transitions can be linked together with those event mechanisms by clicking the icons such as input event or output event show on top of the menu. Then the mechanisms are grouped together according to the control and presentation layers as shown in figure 19. After the marking is initialized, the simulator is ready to run. This simulator has two running modes. The first mode runs step by step, which means it fires all enabled transitions once and waits for the next execution. The second mode runs and fires till no transition is enabled. The simulator simulates an example: mute operation during an MTV playback as demonstrated in figure 19.

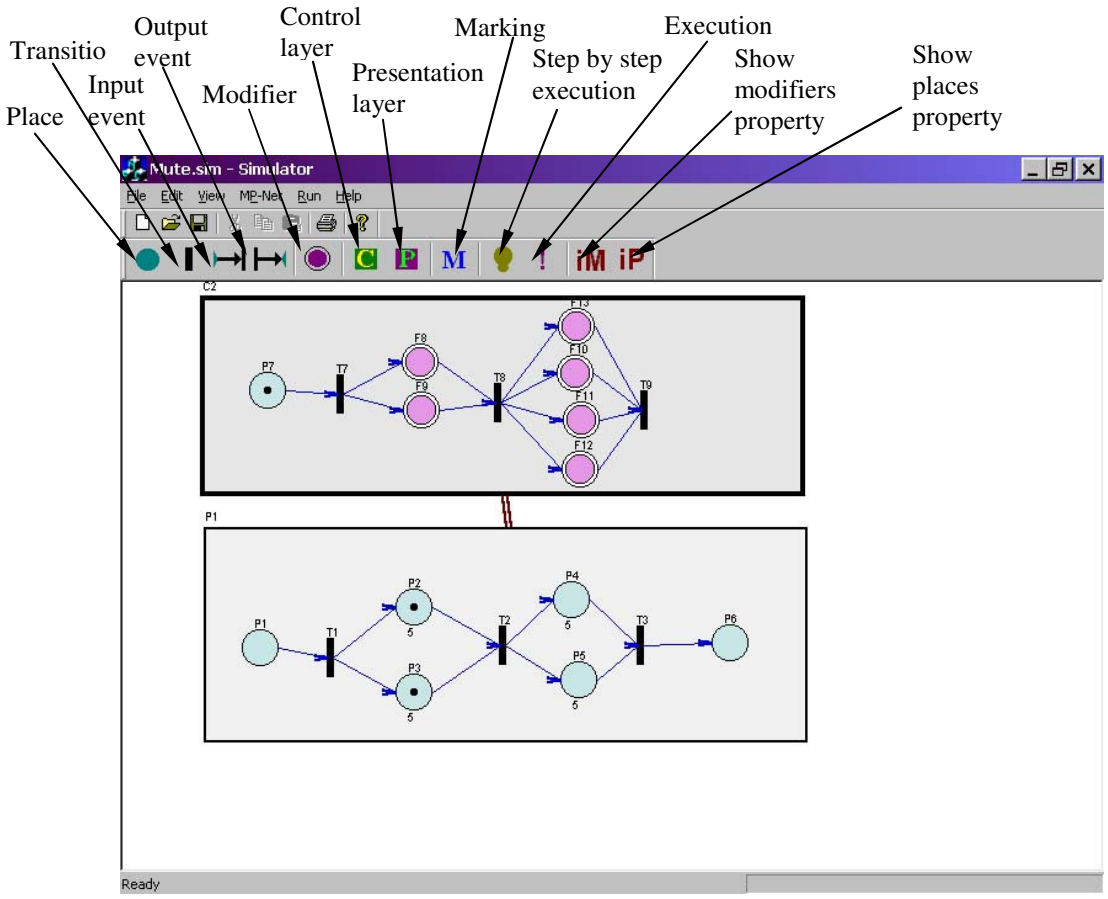


Figure 19. RPN Simulator: Mute operation during a mini MTV playback

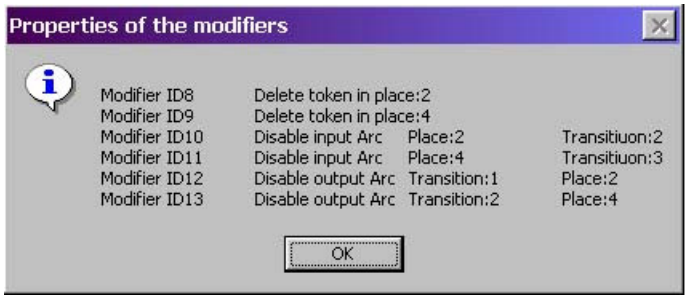


Figure 20. Dialog box of modifiers (see figure 19) property

Each place has a local timer. The timer is initialized to a duration value when the presentation starts. The runtime executive in the simulator periodically updates the timer value associated with each active place. For example, places, p_2, p_3, p_4 and p_5 are associated with duration 5 seconds, place p_7 is associated with duration 2 seconds and places p_1 and p_6 have no duration as indicated in figure 15. If any of these places contain a token (locked), and the user runs the simulator, the duration will count down until zero. Upon the duration reaches zero the token in the place is unlocked and ready to be removed if its transition fires. On the other hand, the token in the modifier is ready to be removed only after its command is executed and then if its transition fires.

Figures 20 and 21 show what happen when the icons "iM" and "iP" respectively are clicked. A dialog box pops up on the screen and this box indicates the legend of the modifiers or places as illustrated in figure 19. In figure 20, it shows that modifiers f_8 and f_9 are commanded to delete tokens contained in places p_2 and p_4, modifiers f_{10} and f_{11} are commanded to disable the input arcs p_2t_2 and p_4t_3 and modifiers f_{12} and f_{13} are commanded to disable the output arcs t_1p_2 and t_2p_4. The legend of the places in figure 19 is self-explained by figure 21.

Once the user clicks on the icon "!" (Execution) on top of the menu, the simulation starts to run. Figure 22 shows a snap -shot of the simulation 3 seconds later after the user initialized the simulator to run. At this instant, the modifiers have executed their commands. Therefore, the programmed tokens, input arcs and output arcs are deleted and disabled respectively. In other words, the model has simulated a mute mode of a mini MTV playback.

Figure 21. Dialog box of places (see figure 19) property

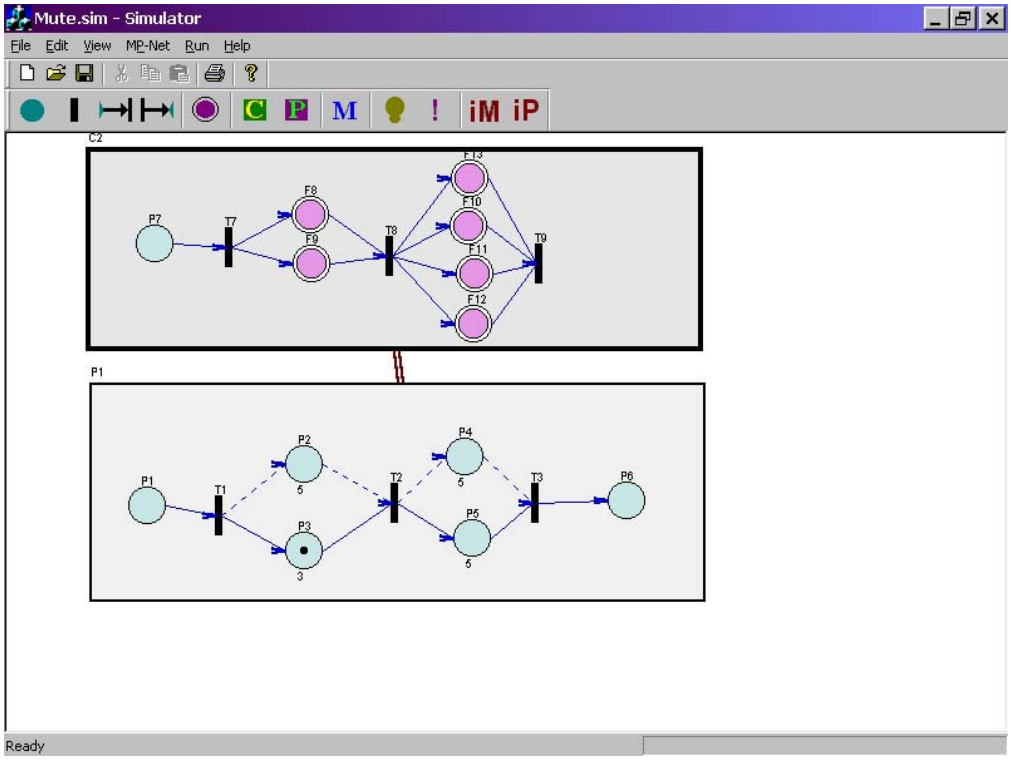


Figure 22. RPN simulation: After the modifiers executed their commands

7. Conclusions and Future Work

We have proposed a powerful mechanism RPN for self-modifying protocol modeling and synchronization control for adaptable multimedia where schedule changes can be made into the presentation layer at run-time. Moreover, we have proved that RPN and its variation (S-RPN) are as powerful as Turing machines. We have also shown that RPN can reduce modeling size. With the comprehensive commands that can be associated with a modifier, the modeling power of RPN is much greater than the conventional Petri net and its extensions in terms of modeling self-modifying

actions and user interactions. Some of the basic user interactions such as zooming, *reverse*, *skip*, *freeze and restart*, and *speed scaling* are modeled. Besides, we have also shown how RPN can be used to model self-modifying ABP.

RPN facilitates the compact and flexible specification of run-time, large-scale specifications while preserving fine granularity as well as supporting real-time user interactions in distributed environment. In conclusion, we have included both the model and theory required to establish the technique's validity and a simulation, which is developed using Visual C++ in Window NT, to show that RPN is feasible.

References

[1] Chang-Li Lin and Sheng-Uei Guan, "The Design and Architecture of a Video Library System", IEEE Communications (Featured Topic Issue on Enterprise Networking), 86-19, Vol. 34, No. 1, Jan. 1996.

[2] Fabio Bastian and Patrick Lenders, “Media Synchronization on Distributed Multimedia Systems”, International Conference on Multimedia Computing and System, pp. 526-531, 1994.

[3] Gerold Blakowski and Ralf Steinmetz, “A Media Synchronization Survey: Reference Model, Specification, and Case Studies”, IEEE Journal on Selected Areas in Communication, Vol. 14, No. 1, pp. 5-35, Jan. 1996.

[4] Chung-Ming Huang and Chung-Ming Lo, “An EFSM-Based Multimedia Synchronization Model and the Authoring System”, IEEE Journal on Selected Areas in Communication, Vol. 14, No. 1, pp. 138-152, Jan. 1996.

[5] Chung-Ming Huang and Chung-Ming Lo, “Synchronization for Interactive Multimedia Presentations”, IEEE Multimedia, Vol. 5, No. 4, pp. 44-62, 1998.

[6] Prabhat K., Andleigh and Kiran T., “Multimedia Systems Design”, Prentice Hall PTR, pp. 421-444, 1996.

[7] Sheng-Uei Guan, Yu Hsiao-Yeh, and Yang Jen-Shun, “A Prioritized Petri Net Model and Its Application in Distributed Multimedia Systems”, IEEE Transactions on Computers, Vol. 47, No. 4, pp. 477-481, Apr. 1998.

[8] Chung-Ming Huang, Chian Wang, and Cheng-Yi Kuo, “A Master-Medium-based Interactive Synchronization Control Scheme for Distributed Multimedia Systems, Euromicro Conference proceedings, Vol. 2, pp. 506-513, 1998.

[9] Bulterman, D.C.A, “SMIL 2.0.2. Examples and Comparisons**”**, IEEE Multimedia, Vol. 9, Iss. 1, pp. 74 -84, Jan.-Mar. 2002.

[10] Tsai, W.T., Ramamoorthy, C.V., Tsai, Wei K., Nishiguchi, Osamu, "Adaptive Hierarchical Routing Protocol", IEEE Transactions on Computers, Vol. 38, Issue 8, pp. 1059-1075, August 1989.

[11] Soon M Chung, “Multimedia Information Storage and Management”, Kluwer Academic Publishers, pp. 303-411, 1996.

[12] Bhatti, N.T., Hiltunen, M.A., Schlichting, R.D., Chiu, W., "Coyote: A System for Constructing Fine-grain Configurable Communication Services", ACM Transactions on Computer Systems Vol. 16, No. 4, pp. 321-366, Nov. 1998.

[13] Rothermel, K. and Helbig, T., "An Adaptive Protocol for Synchronizing Media Streams", Multimedia Systems, Vol. 5, Issue 5, pp. 324-336, 1997.

[14] James L. Peterson, “Petri Net Theory and the Modeling of Systems”, Prentice-Hall Inc., 1981.

[15] Thomas D. C. Little, “Synchronization and Storage Models for Multimedia Objects”, IEEE Journal on Selected Areas in Communication Vol. 8, No.3, pp. 413-427, Apr. 1990.

[16] B. Prabhakaran and S. V. Raghavan, “Synchronization Models for Multimedia Presentation with User Participation”, ACM Multimedia Proceedings, pp. 157-166, Aug. 1993.

[17] Brand, D., Zafiropulo, P., “On Communicating Finite-state Machines”, J. Assoc. Comput. Mach. 30, No. 2, 323-342, 1983.

[18] Gouda,M., Manning,E., Yu,Y.T., “On the Progress of Communication between Two Finite State Machines”, Information and Control, 63(3), 1984, 308-320, 1984.

[19] Schultz, G.D., Rose, D.B., West, C.H. and Gray, J.P., "Executable Description and Validation of SNA", IEEE Trans. on Communications, COM-28, 4, 661-667, 1980.

[20] Sheng-Uei Guan and Zhiqiang Jiang, "A New Approach to Implement Self-Modifying Protocols", Proceedings 2000 IEEE International Symposium on Intelligent Signal Processing and Communication Systems (ISPACS2000), 539-544, Hawaii, Nov. 5-8., 2000.

[21] Naveed U. Qazi, Miae Woo and Arif Ghafoor, “A Synchronization and Communication Model for Distributed Multimedia Objects”, Proceedings First ACM International Conference on Multimedia, pp. 147 - 155, Aug. 1993.

[22] Michel Diaz and Patrick Senac, “Time Stream Petri Nets A Model for Multimedia Streams Synchronization”, Proceedings of the First International Conference on Multimedia Modelling, pp. 257-273, 1993.

[23] Sheng-Uei Guan and Sok-Seng Lim, "Modeling Multimedia with Enhanced Prioritized Petri Nets", Computer Communications, 812-824, Vol. 25, Issue 8, May. 2002.

[24] T. Agerwala, "A Complete Model for Representing the Coordination for Asynchronous Processes", Hopkins Computer Research Report Number 32, Computer Science Program, Johns Hopkins Univ. Baltimore, MD., Jul. 1974.

[25] Rüdiger Valk, "On the Computational Power of Extended Petri Nets", Mathematical Foundations of Computer Science 1978, Berlin, Heidelberg, New York: Springer-Verlag, pp. 526-535, 1978.

Appendix: Analysis of RPNs

The major strength of a RPN is in the modeling of systems, which may exhibit concurrency, synchronization, interaction and distribution (e.g. multimedia applications with user interactions in a distributed environment) and others as illustrated in sections 4 & 5. The following properties of RPN

have been considered in the following, namely: safeness, boundedness, conservation, and synchronization.

Safeness

Conventionally, a place in the net is safe if the number of tokens in that place never exceeds one. A net is safe if all the places in the net are safe.

A place $p_i \in$ P or a modifier $f_j \in$ P of a RPN's layer, S = {T, , P, A, D, L, COM} with a marking M is safe if $M(p_i)$ or $M(f_j) \leq 1$. A RPN is safe if each place in that net is safe. However this property will not be true if a modifier creates/manipulates a new/existing token inside the presentation layer. Other cases like creates/manipulates a new/existing arc can also break the safeness property.

Safeness is a very important property for modeling multimedia applications. If a place is safe, then the place can be implemented as a process that it is utilizing a resource (e.g. text, audio, video, image and others). Assume a place represents a video playback with a specified duration. Hence, a token (locked) in the place represents that the video is playing. When the token in the place is unlocked and removed represents that the video playback had reached the end.

Boundedness

Safeness is a sub-case of the boundedness property. Boundedness refers to limiting a maximum number of tokens present in a place. Hence, a place with *k*-safe or *k*-bounded means that the number of tokens inside a place cannot exceed an integer *k*.

A place $p_i \in$ P and a modifier $f_j \in$ P of a RPN's layer, S = {T, , P, A, D, L, COM} with a marking M is bounded if $M(p_i)$ and $M(f_j) \leq k$. A RPN is bounded if each place in that net is bounded. The same argument as stated earlier applies here.

Conservation

By classic OCPN definitions, some tokens may represent resources. Conservation is an important property such that tokens representing resources are neither created nor destroyed. One of the methods to achieve this conservation property is to ensure that the total number of tokens in a net remains

constant. For the second method, we can make sure that the number of input events to each transition must be equal to the number of output events.

A RPN may not meet the conservation property. As a matter of fact, we need to create and destroy a resource due to a user interaction. For example, when a user requests for a *reverse* operation, a forward resource will be replaced by a *reversed* resource this may break the conservation property.

Synchronization

For traditional Petri nets, a transition can only fire if all its input places have a token. This type of synchronization mechanism is specified before simulation. As for RPN, since a transition can be created dynamically, this means the system under simulation can create synchronization points at run-time. This would be useful for real-time applications or simulations where such flexibility is desired.